The HTTP parser module
PRINTED MANUAL

© 1999-2008 AGG Software

HTTP parser module

© 1999-2008 AGG Software

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: 25.09.2008

Publisher

AGG Software

Production

© 1999-2008 AGG Software
http://www.aggsoft.com

Table of Contents

Part 1 Introduction 1

Part 2 System requirements 1

Part 3 Installing HTTP parser module 2

Part 4 Glossary 5

Part 5 User Manual 6

1 Parsing data 6
2 Filter 10
3 Syntax of Regular Expressions 11
4 Response 16

Part 6 Troubles? 18

1 Possible problems 18

© 1999-2008 AGG Software

1 Introduction

This parser module mostly works together with Advanced TCP/IP Data Logger and implements very basic HTTP server features. Hypertext Transfer Protocol (HTTP) is a communications protocol for the transfer of information on the Internet. HTTP is a request/response standard between a client and a server. A client is your device, the server is our application. The client making an HTTP request and send any data. Our software parse requests and can export it with many data export modules.

The parser allows you to parse and export variables from a HTTP header and a HTTP request body string and sends answers, if this is necessary. This isn't required to install other HTTP servers and create any scripts for web servers to collect and export data from your devices.

Key features:

- Parsing of a HTTP header;
- Parsing of a query string;
- Detects a request method (POST or GET) and a protocol automatically;
- Allows extract and convert parser items from the string data type to other data types;
- The parser can filter requests;
- Allows you to send responses with code 403, 404 or from a file or a folder.

2 System requirements

The following requirements must be met for "HTTP parser module" to be installed:

Operating system:

- Windows 95;
- Windows 95 OSR2;
- Windows 98;
- Windows Me;
- Windows NT4;
- Windows 2000;
- Windows XP (all editions) (both x86 and x64);
- Windows 2003 Server (both x86 and x64);
- Windows Vista (both x86 and x64).

Free disk space: Not less than 5 MB of free disk space is recommended.
Special access requirements: You should log on as a user with Administrator rights in order to install this module.

The main application (core) must be installed, for example, Advanced Serial Data Logger.

Notes for Microsoft Vista:

Since our software saves data to the registry and installs to the Program Files folder, the following requirements must be met:

1. You need Administrator rights to run and install our software

© 1999-2008 AGG Software

2. The shortcut icon of our software will be located on the desktop;
3. Windows Vista will ask for your confirmation to continue the installation.

NOTE: You can configure the user account only once in order not to see the above dialog box any more. Search Google for the solution of this problem.

3 Installing HTTP parser module

1. Close the main application (for example, Advanced Serial Data Logger) if it is running;
2. Copy the program to your hard drive;
3. Run the module installation file with a double click on the file name in Windows Explorer;
4. Follow the instructions of the installation software. Usually, it is enough just to click the "Next" button several times;
5. Start the main application. The name of the module will appear on the "Modules" tab of the "Settings" window if it is successfully installed.

If the module is compatible with the program, its name and version will be displayed in the module list. You can see examples of installed modules on fig.1-2. Some types of modules require additional configuration. To do it, just select a module from the list and click the "Setup" button next to the list. The configuration of the module is described below.

You can see some types of modules on the "Log file" tab. To configure such a module, you should select it from the "File type" list and click the "Advanced" button.

© 1999-2008 AGG Software

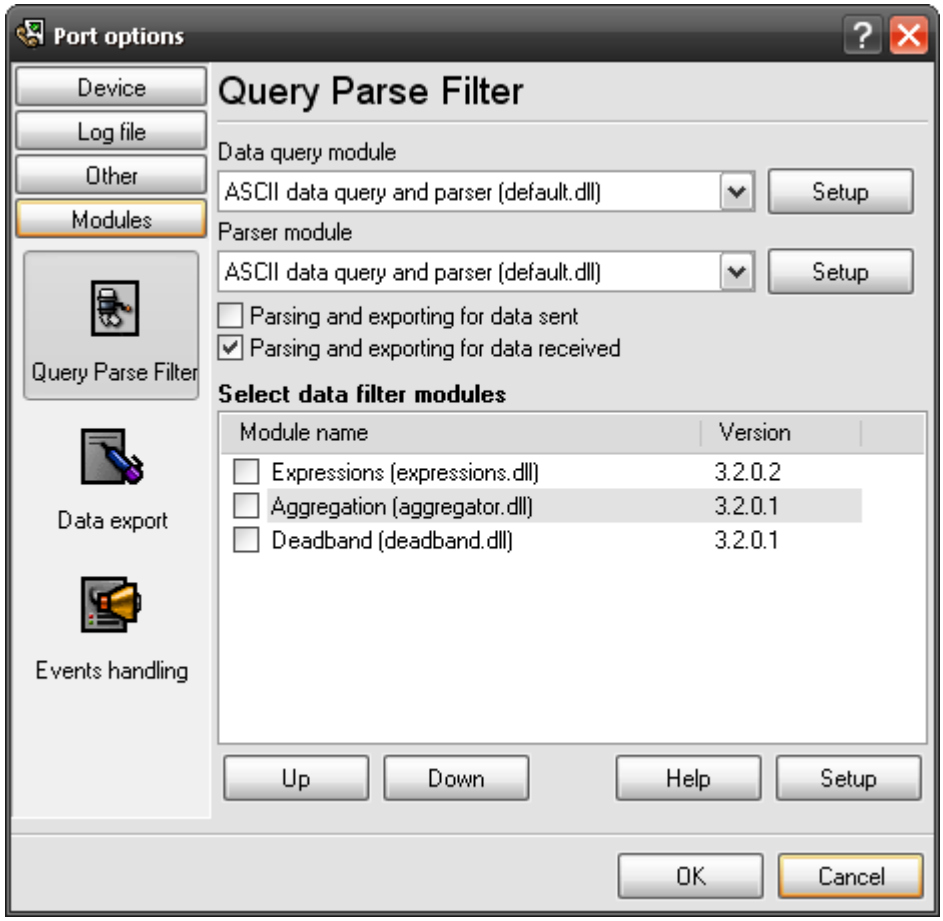

Fig.1. Examples of installed modules

© 1999-2008 AGG Software

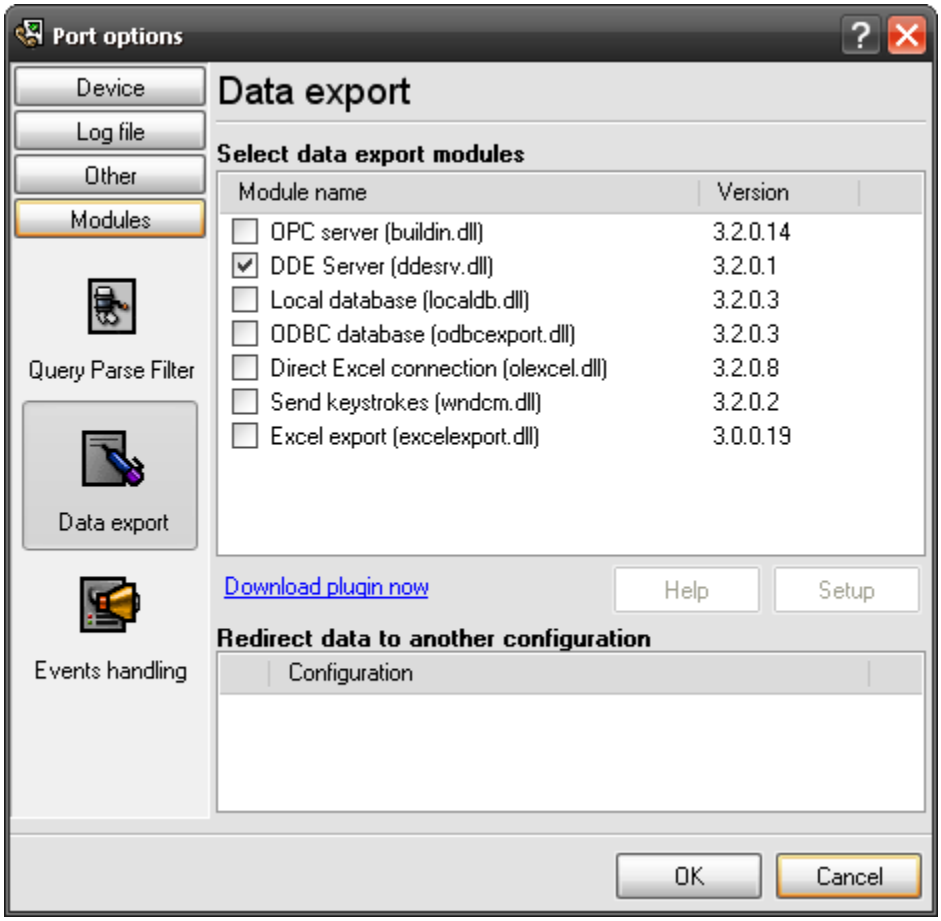

Fig.2. Examples of installed modules

© 1999-2008 AGG Software

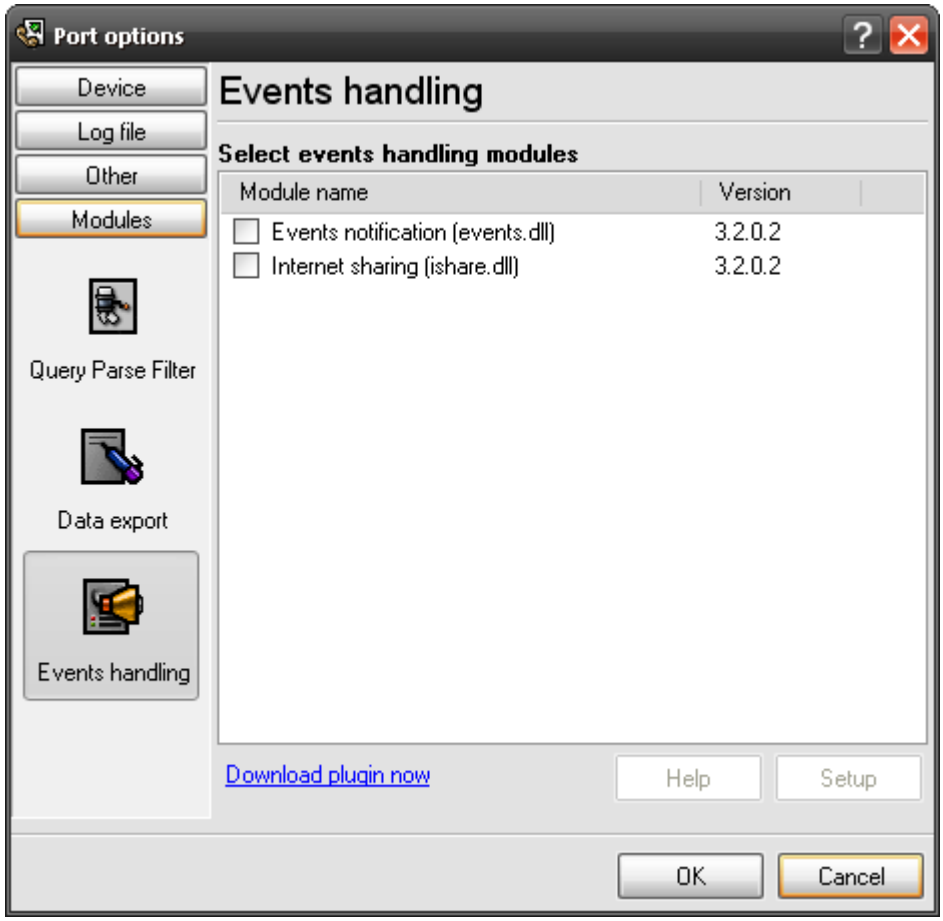

Fig.3. Examples of installed modules

4 Glossary

Plug-in - module

Main program – the program shell that uses this module. For example: Advanced Serial Data Logger

Parser – the module that processes the data flow singling out data packets from it and variables from data packets. These variables are used in data export modules after that.

Core - see "Main program".

© 1999-2008 AGG Software

5 User Manual

5.1 Parsing data

The request message consists of the following:

- Request line, such as GET /images/logo.gif HTTP/1.1, which requests the file logo.gif from the /images directory
- Headers, such as Accept-Language: en
- An empty line
- An optional message body

The request line and headers must all end with <CR><LF> (that is, a carriage return followed by a line feed). The empty line must consist of only <CR><LF> and no other whitespace. In the HTTP/1.1 protocol, all headers except Host are optional.

The parser verifies an HTTP message and extracts all fields from a header or message body. The parser doesn't work with body of multipart/mixed POST requests.

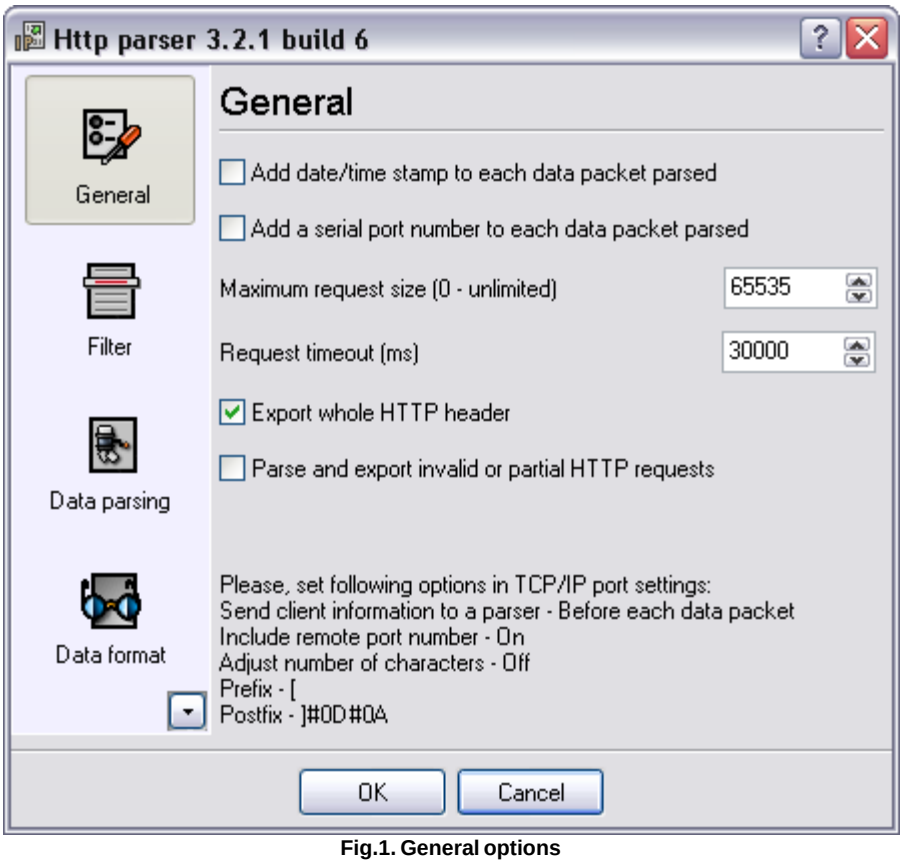
Http parser 3.2.1 build 6
General
Filter
Data parsing
Data format
General
Add date/time stamp to each data packet parsed
Add a serial port number to each data packet parsed
Maximum request size (0 - unlimited)
65535
Request timeout (ms)
30000
Export whole HTTP header
Parse and export invalid or partial HTTP requests
Please, set following options in TCP/IP port settings:
Send client information to a parser - Before each data packet
Include remote port number - On
Adjust number of characters - Off
Prefix - [
Postfix -]#0D#0A
OK
Cancel

Fig.1. General options

- **Add a time stamp to each data packet** – the parser will add the new "data stamp" variable to each data packet that will be parsed into variables;
- **Add a port number to each data packet** – the parser will add the new "port number" variable to each data packet that will be parsed into variables; It will allow you to identify data during export if you collect data from several devices at the same time.

© 1999-2008 AGG Software

- **Maximum request size** – you can specify the size of the request. This options allows you to protect the server from bad large request messages. If this values is greater than zero, then the parser will parse first N bytes only;
- **Request timeout** – if the parser didn't receive a valid HTTP request from the client within this interval, then the parser may ignore this request and don't export data;
- **Export whole HTTP header** – if this option is on, then the parser will parse and export all header items. Otherwise the parser will export, which you'll define on the "Data parsing" page. If you'll define the variable on the "Data parsing" page, then you can additionally convert header items to any data type;
- **Parse and export invalid or partial HTTP requests** – if this option is on, then the parser will parse and export HTTP requests, which don't match the HTTP specification or received partially.

Data parsing variants

You can find several real-life examples on our site where different data parsing methods are used. It can help you understand how the parser works.
http://www.aggsoft.com/serial-data-logger/tutorials.htm

Example:

Please take a look at the example below (it is a typical HTTP request):

```
GET /js/some.js HTTP/1.1
Host: 127.0.0.1:10000
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; rv:1.8.1.17) Gecko/20080829 Firefox/2.0.0.17
Accept: */*
Accept-Language: q=0.8,en-us;q=0.5,en;q=0.3
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://127.0.0.1:10000/incl/meta1.txt
```

The data above consists of several strings with different length, all values in the string are delimited by :. In our case, they are:

- **Host** - the string "127.0.0.1:10000";
- **User-Agent** – user agent;
- and so on

The parameter window is located on the "Data parsing" page (fig.2). You may add several variables and specify their parameters and a data type: string, date, time, etc. Each variable in the list may have a different parsing type. Data types supported by the module are listed below.

© 1999-2008 AGG Software

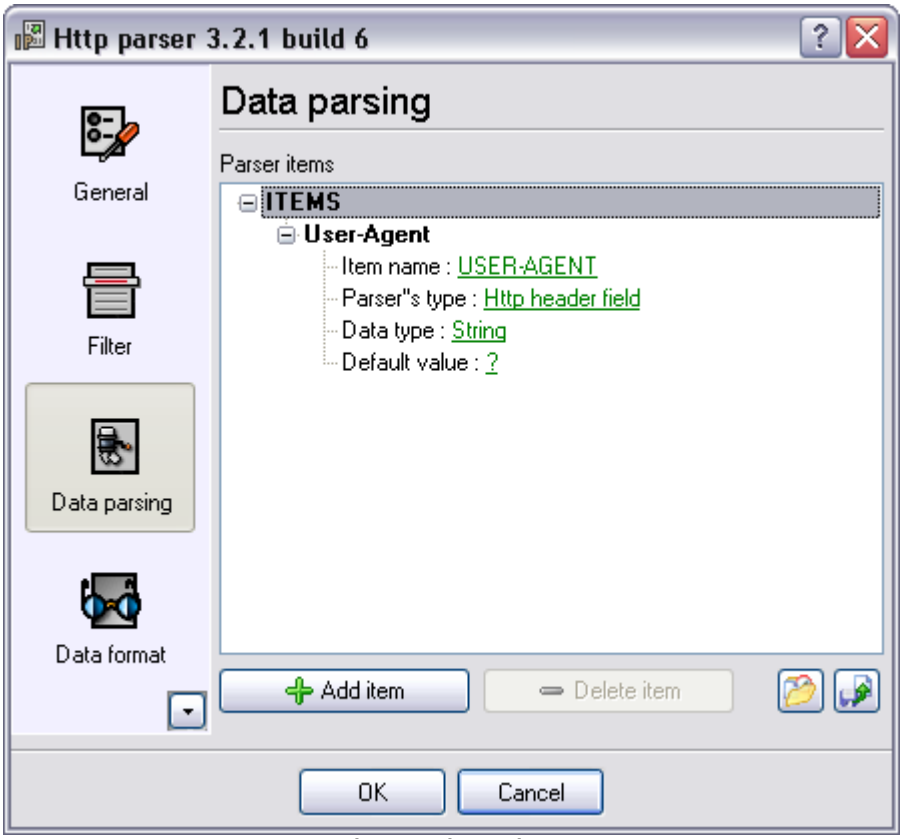

Fig.2. Parsing options

1. Http header field

The parser will extract a field from a header with name, which you'll define in the "Item name" field. Additionally you can specify the data type and default value for this field.

2. Query string field

Same as above, but works with an HTTP request body (query string).

3. Fixed position

Allows you to extract data from a fixed position in your HTTP requests. It is useful if you want to extract the method of a HTTP request (POST or GET).

4. Delimited

Allows you to extract values, which are delimited by any character or text. In our case, you can extract header field using the ": " delimiter.

5. None – If you select this type, the parser will not extract a value from the data packet, but will just create a variable with the NULL value.

6. Default – If you select this type, the parser will not single out a value from the data packet, but will just create a variable with the value you specify in the "Default value" field.

© 1999-2008 AGG Software

7. **Regular expression** - a widely used data parsing method that uses templates with certain syntax to search for data. The general rules of specifying regular expressions are described in next chapter. In this mode you must specify regular expression text with one subexpression. Result of this subexpression will be used as parsing result.

Data formatting

You can use the following tab to specify the data format for some data types (fig.3).

Fig.3. Data format

Text data format – this group of options allows you to remove leading and/or trailing spaces in variables of the String type.

Date and time format – it often happens that the date or time format the external device sends does not coincide with the format in which this data is stored in the database. To convert the date into the necessary format, you should specify the format the date and time is received in.

The same format is used to specify the date and time as the one that is used in the name of the log file in the main program (for example, Advanced Serial Data Logger). So read about it in the help file of the main program. Note: the "/" and/or ":" characters are enclosed in quotation marks in the template (see fig.4).

If some part is missing in the received date and time (for example, year), you can take the missing part from the current date and time:

© 1999-2008 AGG Software

Use the current date for missing parts;
Use the current time for missing parts;

Sometimes devices send dates with month names (for example, "Jan, 10 2005") and this name can be in a language (in this example, it is in English) different from the language of your operating system. The language ID the date is sent in is specified in the "**Date language ID**" field in this case.

Data types:

- String - String value - Character array with length from 1 to 65535 characters;
- Memo - String value - Character array with length from 1 to 2^32 characters;
- Bytes - Binary value;
- Blob - Binary Large Object field (bytes set);
- Boolean - Logical value (True/False) - 0 or 1;
- Float - Real number - value range: -2.9 x 10^-39 .. 1.7 x 10^38
- Smallint - Signed small integer - value range: 32768..32767;
- Word - Word (unsigned small integer) - value range: 0..65535;
- Integer - Signed integer value: -2147483648..2147483647;
- Date - Date;
- Time - Time;
- DateTime - Date and time.

5.2 Filter

The filter is used to ignore some data packets that you do not want to export with the help of other modules.

Fig.1 Filtering rules

You should define one or several filtering rules. If some condition is true, the program does what

© 1999-2008 AGG Software

is specified in the "Action" field with the matching packet.

Action types

- **Ignore** – the current data packet will be ignored and will not be exported;
- **Parse** – the current data packet will be parsed and exported.

There are several condition types that are specified in the "Status" field.

Rule status types

- **Disabled** – this rule is disabled and not used for filtering purposes;
- **Contains** – this rules checks whether the string/expression from the "Expression" field is present in the data packet;
- **Does not contain** – this rules checks whether the string/expression from the "Expression" field is absent in the data packet.

Expressions types – Expressions in the "Expression" field can be of 2 types:

- **Text** – the module will search the data packet for a string specified in the "Expression" field. The search is case-sensitive.
- **Regular expression** – the module will use the regular expression specified in the "Expression" field in its search. The search is case-sensitive.

5.3 Syntax of Regular Expressions

Introduction

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special **metacharacters** allow You to specify, for instance, that a particular string You are looking for occurs at the beginning or end of a line, or contains **n** recurrences of a certain character.

Regular expressions look ugly for novices, but really they are very simple, handy and powerful tool.

Let's start our learning trip!

Simple matches

Any single character matches itself, unless it is a **metacharacter** with a special meaning described below.

A series of characters matches that series of characters in the target string, so the pattern "bluh" would match "bluh" in the target string. Quite simple, eh ?

You can cause characters that normally function as **metacharacters** or **escape sequences** to be interpreted literally by 'escaping' them by preceding them with a backslash "\", for instance: metacharacter "^" match beginning of string, but "\^" match character "^", "\\" match "\" and so on.

© 1999-2008 AGG Software

Examples:

foobar	*matchs string 'foobar'*
\^FooBarPtr	*matchs '^FooBarPtr'*

Escape sequences

Characters may be specified using a **escape sequences** syntax much like that used in C and Perl: "\n" matches a newline, "\t" a tab, etc. More generally, \xnn, where nn is a string of hexadecimal digits, matches the character whose ASCII value is nn. If You need wide (Unicode) character code, You can use '\x{nnnn}', where 'nnnn' - one or more hexadecimal digits.

\xnn	*char with hex code nn*
\x{nnnn}	*char with hex code nnnn (one byte for plain text and two bytes for Unicode)*
\t	*tab (HT/TAB), same as \x09*
\n	*newline (NL), same as \x0a*
\r	*car.return (CR), same as \x0d*
\f	*form feed (FF), same as \x0c*
\a	*alarm (bell) (BEL), same as \x07*
\e	*escape (ESC), same as \x1b*

Examples:

foo\x20bar	*matchs 'foo bar' (note space in the middle)*
\tfoobar	*matchs 'foobar' predefined by tab*

Character classes

You can specify a **character class**, by enclosing a list of characters in [], which will match any **one** character from the list.

If the first character after the "[" is "^", the class matches any character **not** in the list.

Examples:

foob[aeiou]r	*finds strings 'foobar', 'foober' etc. but not 'foobbr', 'foobcr' etc.*
foob[^aeiou]r	*find strings 'foobbr', 'foobcr' etc. but not 'foobar', 'foober' etc.*

Within a list, the "-" character is used to specify a **range**, so that a-z represents all characters between "a" and "z", inclusive.

If You want "-" itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash. If You want ']' you may place it at the start of list or escape it with a backslash.

Examples:

[-az]	*matchs 'a', 'z' and '-'*
[az-]	*matchs 'a', 'z' and '-'*
[a\-z]	*matchs 'a', 'z' and '-'*
[a-z]	*matchs all twenty six small characters from 'a' to 'z'*
[\n-\x0D]	*matchs any of #10,#11,#12,#13.*
[\d-t]	*matchs any digit, '-' or 't'.*
[]-a]	*matchs any char from ']'..'a'.*

© 1999-2008 AGG Software

Metacharacters

Metacharacters are special characters which are the essence of Regular Expressions. There are different types of metacharacters, described below.

Metacharacters - line separators

^	*start of line*
$	*end of line*
\A	*start of text*
\Z	*end of text*
.	*any character in line*

Examples:

^foobar	*matchs string 'foobar' only if it's at the beginning of line*
foobar$	*matchs string 'foobar' only if it's at the end of line*
^foobar$	*matchs string 'foobar' only if it's the only string in line*
foob.r	*matchs strings like 'foobar', 'foobbr', 'foob1r' and so on*

The "^" metacharacter by default is only guaranteed to match at the beginning of the input string/text, the "$" metacharacter only at the end. Embedded line separators will not be matched by "^" or "$".
You may, however, wish to treat a string as a multi-line buffer, such that the "^" will match after any line separator within the string, and "$" will match before any line separator.

The "." metacharacter by default matches any character.

Note that "^.*$" (an empty line pattern) doesnot match the empty string within the sequence \x0D\x0A, but matchs the empty string within the sequence \x0A\x0D.

Metacharacters - predefined classes

\w	*an alphanumeric character (including "_")*
\W	*a nonalphanumeric*
\d	*a numeric character*
\D	*a non-numeric*
\s	*any space (same as [\t\n\r\f])*
\S	*a non space*

You may use \w, \d and \s within custom **character classes**.

Examples:

foob\dr	*matchs strings like 'foob1r', "foob6r' and so on but not 'foobar', 'foobbr' and so on*
foob[\w\s]r	*matchs strings like 'foobar', 'foob r', 'foobbr' and so on but not 'foob1r', 'foob=r' and so on*

Metacharacters - iterators

Any item of a regular expression may be followed by another type of metacharacters - **iterators**. Using this metacharacters You can specify number of occurences of previous character, **metacharacter** or **subexpression**.

© 1999-2008 AGG Software

*	*zero or more ("greedy"), similar to {0,}*
+	*one or more ("greedy"), similar to {1,}*
?	*zero or one ("greedy"), similar to {0,1}*
{n}	*exactly n times ("greedy")*
{n,}	*at least n times ("greedy")*
{n,m}	*at least n but not more than m times ("greedy")*
*?	*zero or more ("non-greedy"), similar to {0,}?*
+?	*one or more ("non-greedy"), similar to {1,}?*
??	*zero or one ("non-greedy"), similar to {0,1}?*
{n}?	*exactly n times ("non-greedy")*
{n,}?	*at least n times ("non-greedy")*
{n,m}?	*at least n but not more than m times ("non-greedy")*

So, digits in curly brackets of the form {n,m}, specify the minimum number of times to match the item n and the maximum m. The form {n} is equivalent to {n,n} and matches exactly n times. The form {n,} matches n or more times. There is no limit to the size of n or m, but large numbers will chew up more memory and slow down r.e. execution.

If a curly bracket occurs in any other context, it is treated as a regular character.

Examples:

foob.*r	*matchs strings like 'foobar', 'foobalkjdflkj9r' and 'foobr'*
foob.+r	*matchs strings like 'foobar', 'foobalkjdflkj9r' but not 'foobr'*
foob.?r	*matchs strings like 'foobar', 'foobbr' and 'foobr' but not 'foobalkj9r'*
fooba{2}r	*matchs the string 'foobaar'*
fooba{2,}r	*matchs strings like 'foobaar', 'foobaaar', 'foobaaaar' etc.*
fooba{2,3}r	*matchs strings like 'foobaar', or 'foobaaar' but not 'foobaaaar'*

A little explanation about "greediness". "Greedy" takes as many as possible, "non-greedy" takes as few as possible. For example, 'b+' and 'b*' applied to string 'abbbbc' return 'bbbb', 'b+?' returns 'b', 'b*?' returns empty string, 'b{2,3}?' returns 'bb', 'b{2,3}' returns 'bbb'.

Metacharacters - alternatives

You can specify a series of **alternatives** for a pattern using "|" to separate them, so that fee|fie|foe will match any of "fee", "fie", or "foe" in the target string (as would f(e|i|o)e). The first alternative includes everything from the last pattern delimiter ("(", "[", or the beginning of the pattern) up to the first "|", and the last alternative contains everything from the last "|" to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end.
Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching foo|foot against "barefoot", only the "foo" part will match, as that is the first alternative tried, and it successfully matches the target string. (This might not seem important, but it is important when you are capturing matched text using parentheses.)
Also remember that "|" is interpreted as a literal within square brackets, so if You write [fee|fie|foe] You're really only matching [feio|].

Examples:

foo(bar|foo)	*matchs strings 'foobar' or 'foofoo'.*

© 1999-2008 AGG Software

Metacharacters - subexpressions

The bracketing construct (...) may also be used for define r.e. subexpressions.

Subexpressions are numbered based on the left to right order of their opening parenthesis. First subexpression has number '1'

Examples:

(foobar){8,10}	*matchs strings which contain 8, 9 or 10 instances of the 'foobar'*
foob([0-9]|a+)r	*matchs 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' etc.*

Metacharacters - backreferences

Metacharacters \1 through \9 are interpreted as backreferences. \<n> matches previously matched **subexpression** #<n>.

Examples:

(.)\1+	*matchs 'aaaa' and 'cc'.*
(.+)\1+	*also match 'abab' and '123123'*

(["']?)(\d+)\1 matchs '"13" (in double quotes), or '4' (in single quotes) or 77 (without quotes) etc

Modifiers

Modifiers are for changing behaviour of parser.

There are many ways to set up modifiers.
Any of these modifiers may be embedded within the regular expression itself using the (?...) construct.

i
Do case-insensitive pattern matching (using installed in you system locale settings).

m
Treat string as multiple lines. That is, change "^" and "$" from matching at only the very start or end of the string to the start or end of any line anywhere within the string.

s
Treat string as single line. That is, change "." to match any character whatsoever, even a line separators, which it normally would not match.

g
Non standard modifier. Switching it Off You'll switch all following operators into non-greedy mode (by default this modifier is On). So, if modifier /g is Off then '+' works as '+?', '*' as '*?' and so on

x
Extend your pattern's legibility by permitting whitespace and comments (see explanation below).

The modifier /x itself needs a little more explanation. It tells the parser to ignore whitespace that is neither backslashed nor within a character class. You can use this to break up your regular expression into (slightly) more readable parts. The # character is also treated as a metacharacter introducing a comment, for example:

(

© 1999-2008 AGG Software

```
  (abc) # comment 1
| # You can use spaces to format r.e. - parser ignores it
  (efg) # comment 2
  )
```

This also means that if you want real whitespace or # characters in the pattern (outside a character class, where they are unaffected by /x), that you'll either have to escape them or encode them using octal or hex escapes. Taken together, these features go a long way towards making regular expressions text more readable.

How to change modifiers

(?imsxr-imsxr)
You may use it into r.e. for modifying modifiers by the fly. If this construction inlined into subexpression, then it effects only into this subexpression

Examples:

(?i)New-York	*matchs 'New-york' and 'New-York'*
(?i)New-(?-i)York	*matchs 'New-York' but not 'New-york'*
(?i)(New-)?York	*matchs 'New-york' and 'new-york'*
((?i)New-)?York	*matchs 'New-York', but not 'new-york*

(?#text)
A comment, the text is ignored. Note that parser closes the comment as soon as it sees a ")", so there is no way to put a literal ")" in the comment.

5.4 Response

Upon receiving the request, the module may send back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested file, an error message, or some other information. **Content type** specifies the Internet media type of the data conveyed by the http message.

© 1999-2008 AGG Software

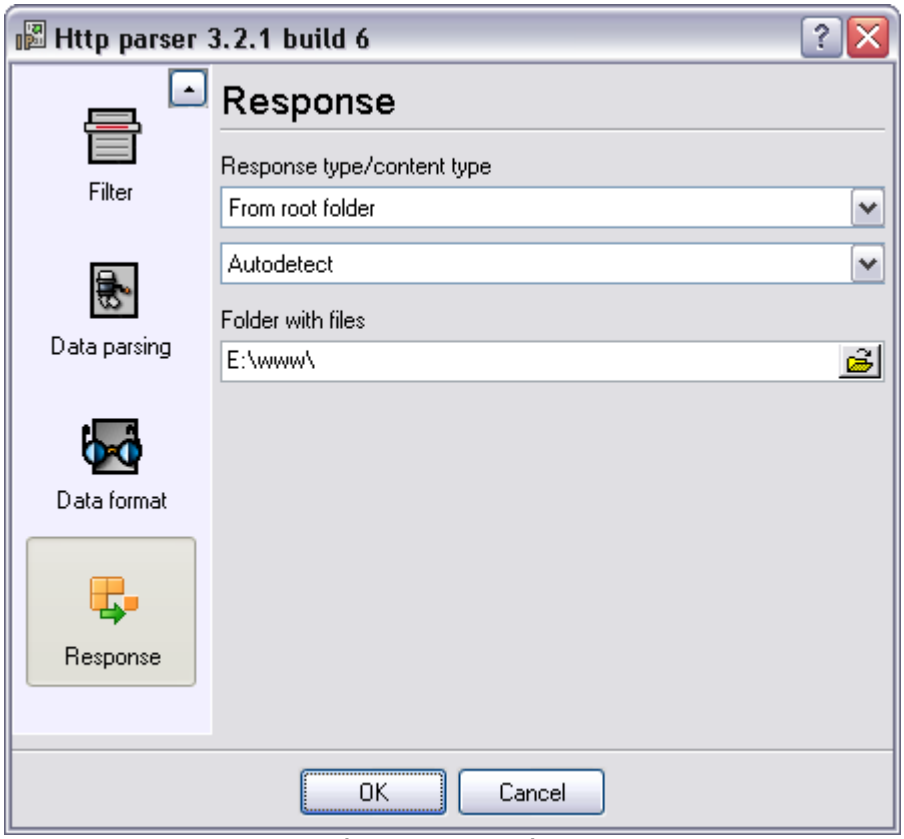

Fig.5. Response options

The module allows supports following response types:

- **Disabled** - the module will not send responses at all. Please, note, that most HTTP clients will wait any response;
- **403** - the module will always send a response message with the 403 (Forbidden) status code;
- **404** - the module will always send a response message with the 404 (Not Found) status code;
- **Text** - you may define any plain text or HTML in the corresponding field. The module will send it with the 200 (OK) status code;
- **From file** - the module may read response from a text or binary file and send it with the 200 (OK) status code. The module will ignore a file name, which is specified in the HTTP request. If the file doesn't exist then the module will return the 404 message;
- **From root folder** - the module extracts a file name from the HTTP request and search this file in the specified folder. If the file exists, then module send it with the 200 (OK) status code. Otherwise the module will return the 404 message. We recommend to use the "**Autodetect**" content type for this response type;

© 1999-2008 AGG Software

6 Troubles?

6.1 Possible problems

No data for publication/exporting – no data is passed for exporting. Solution: configure the parser, make sure that one or more variables are declared in the parser.

Error on binding variable with name %s [%s] – the error usually occurs if data does not correspond to the specified format. For example, the date and time format does not correspond to the data.

Unable to disconnect from the database [%s] and **Unable to connect to a database [%s]** – it is impossible to connect/disconnect to/form the database. You should check the parameters of the database connection. The analysis of the additional information will help you locate the error.

Database access error [%s]. Stop operations with the database? – the message appears if an error occurs during an attempt to execute an SQL query if the second variant of reacting to errors is selected. The message implies a "Yes" or "No" answer. The analysis of the additional information will help you locate the error.

Unable to verify your SQL script [%s] – the message appears when an attempt to analyze your SQL query fails. Check if the syntax of your SQL query is correct.

Tested successfully – the message appears if your database connection is successfully tested. It requires no additional actions.

Database isn't used – the message appears if the module is temporarily disabled (the "Temporarily disabled" check box is selected) or the database name field is empty. Check the connection parameters.

Database isn't selected - the message appears if the database type is not selected. Check the connection parameters.

Database: %s – %s contains the database name. The message appears if the database connection is successful. Usually, you see it when you call the module for the first time. It requires no additional actions.

Invalid data block length (columns=%d,length=%d) – an internal application error. It means that the data sent by the parser is in an invalid format. Perhaps, you are using the module incompatible with the version of the Advanced Serial Data Logger kernel. Update the versions of both the kernel and the module.

The time of connection is not due yet (%d,%d) – the message appears during an attempt to connect to the database after the connection to it has been lost and the "Reconnect after" option is enabled. No additional actions are required.

Invalid procedure call. Bad arguments –an attempt to call the module using invalid parameters. Perhaps, you are using the module incompatible with the version of the Advanced Serial Data Logger kernel. Update the versions of both the kernel and the module.

Writing to the database is complete - the message appears if your queue of SQL queries is successfully executed. It requires no additional actions.

© 1999-2008 AGG Software

Writing to the database is complete with errors – the message appears if the executing your queue of SQL queries was interrupted by an error. It requires no additional actions.

Your SQL is empty. Please, specify some SQL text first – the message appears if you do not enter the text for your SQL query. Check if the options on the "SQL queue" tab are configured correctly.

Invalid temporary path – the path to the temporary file specified by you does not exist. Enter a new path in the "Temporary folder" field on the "Errors handling" tab.

%s, %d – will be replaced by additional information.

© 1999-2008 AGG Software